Explainable Network Growth by Folding on Typical Mis-predictions

Jimmy KANG

March 13, 2025

Problem Statement

- The training of artificial deep neural networks (DNNs) has long suffered from their **enormous scale**.

Problem Statement

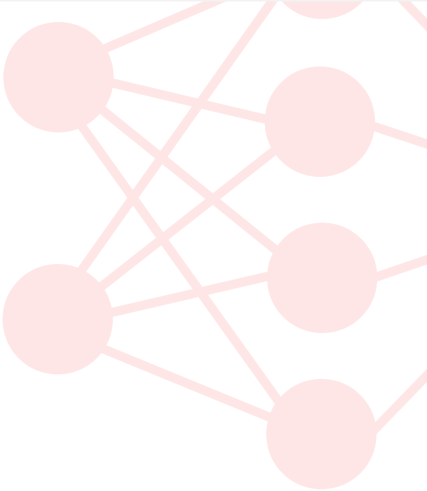
- The training of artificial deep neural networks (DNNs) has long suffered from their **enormous scale**.
- The black box nature of the training process weakens the **interpretability** of neural network models in understanding of the mechanisms behind their decision-making.

Problem Statement

- The training of artificial deep neural networks (DNNs) has long suffered from their **enormous scale**.
- The black box nature of the training process weakens the **interpretability** of neural network models in understanding of the mechanisms behind their decision-making.
- The existing training strategy initializes **over-parameterized** models and allows representations to be learned in an anarchic manner from large amounts of data.
  - found to **underutilize the approximability** of the model
  - **decision criteria** are arranged in an unordered and susceptible manner

Related works

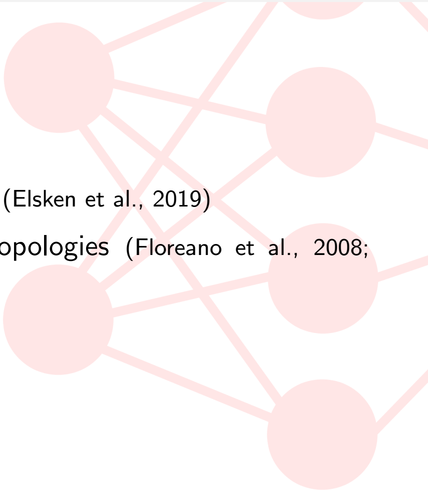
1. Network pruning (Cheng et al., 2024)



Related works

1. Network pruning (Cheng et al., 2024)
2. Neural Architecture Search (NAS) (Elsken et al., 2019)

Related works



1. Network pruning (Cheng et al., 2024)
2. Neural Architecture Search (NAS) (Elsken et al., 2019)
3. NeuroEvolution of Augmenting Topologies (Floreano et al., 2008; Galván and Mooney, 2021)

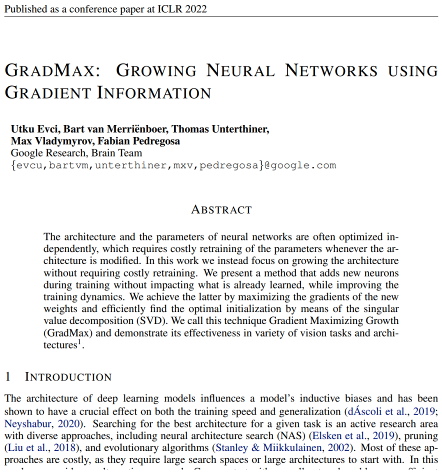
Related works

1. Network pruning (Cheng et al., 2024)
2. Neural Architecture Search (NAS) (Elsken et al., 2019)
3. NeuroEvolution of Augmenting Topologies (Floreano et al., 2008; Galván and Mooney, 2021)
4. Scalable Deep Learning / Dynamic Neural Network (Han et al., 2021)

Related works

1. Network pruning (Cheng et al., 2024)
2. Neural Architecture Search (NAS) (Elsken et al., 2019)
3. NeuroEvolution of Augmenting Topologies (Floreano et al., 2008; Galván and Mooney, 2021)
4. Scalable Deep Learning / Dynamic Neural Network (Han et al., 2021)
5. ...

Baseline: GradMax (Evci, et al., 2022)



Published as a conference paper at ICLR 2022

GRADMAX: GROWING NEURAL NETWORKS USING GRADIENT INFORMATION

Utku Evci, Bart van Merriënboer, Thomas Unterthiner,
Max Vladymyrov, Fabian Pedregosa
Google Research, Brain Team
{evcu,bartvm,unterthiner,mxv,pedregosa}@google.com

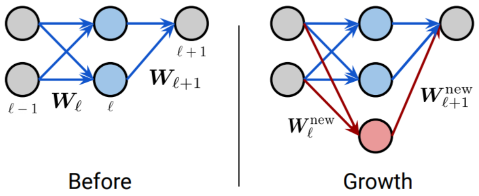
ABSTRACT

The architecture and the parameters of neural networks are often optimized independently, which requires costly retraining of the parameters whenever the architecture is modified. In this work we instead focus on growing the architecture without requiring costly retraining. We present a method that adds new neurons during training without impacting what is already learned, while improving the training dynamics. We achieve the latter by maximizing the gradients of the new weights and efficiently find the optimal initialization by means of the singular value decomposition (SVD). We call this technique Gradient Maximizing Growth (GradMax) and demonstrate its effectiveness in variety of vision tasks and architectures[1].

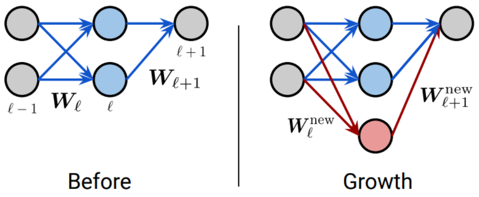
1 INTRODUCTION

The architecture of deep learning models influences a model's inductive biases and has been shown to have a crucial effect on both the training speed and generalization (dÁscoli et al., 2019; Neyshabur, 2020). Searching for the best architecture for a given task is an active research area with diverse approaches, including neural architecture search (NAS) (Elsken et al., 2019), pruning (Liu et al., 2018), and evolutionary algorithms (Stanley & Miikkulainen, 2002). Most of these approaches are costly, as they require large search spaces or large architectures to start with. In this

Baseline: GradMax (Evci, et al., 2022)



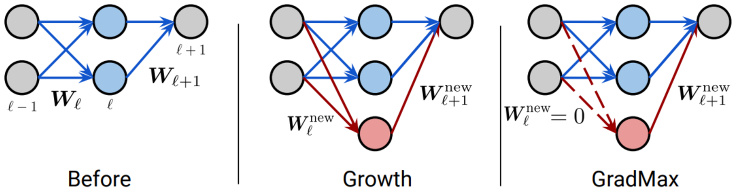
Baseline: GradMax (Evci, et al., 2022)



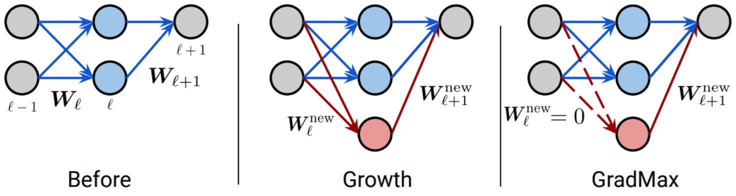
$$\boldsymbol{W}_\ell^+ = \begin{bmatrix} \boldsymbol{W}_\ell \\ \boldsymbol{W}_\ell^{\text{new}} \end{bmatrix}$$

$$\boldsymbol{W}_{\ell+1}^+ = \begin{bmatrix} \boldsymbol{W}_{\ell+1} & \boldsymbol{W}_{\ell+1}^{\text{new}} \end{bmatrix}.$$

Baseline: GradMax (Evci, et al., 2022)

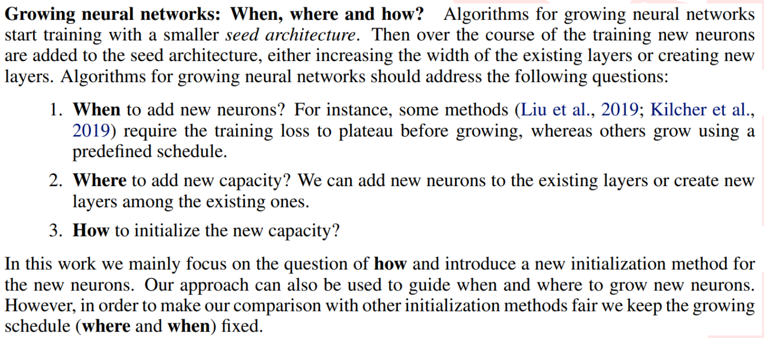


Baseline: GradMax (Evci, et al., 2022)



$$\underset{W_{\ell+1}^{\text{new}}}{\arg\max} \left\| W_{\ell+1}^{\text{new},\top} \mathbb{E}_D \left[\frac{\partial L}{\partial z_{\ell+1}} h_{\ell-1}^\top \right] \right\|_F^2, \quad \text{s.t. } \left\| W_{\ell+1}^{\text{new}} \right\|_F \leq c\,.$$

Baseline: GradMax (Evci, et al., 2022)



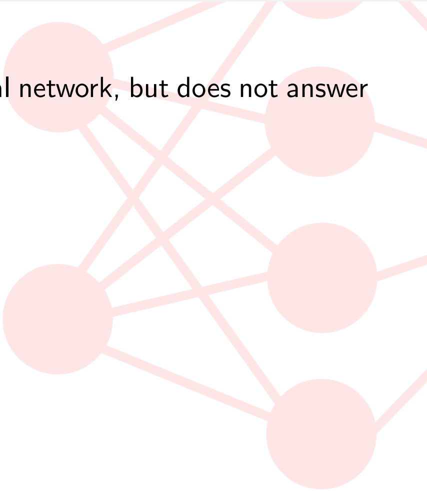
Growing neural networks: When, where and how? Algorithms for growing neural networks start training with a smaller *seed architecture*. Then over the course of the training new neurons are added to the seed architecture, either increasing the width of the existing layers or creating new layers. Algorithms for growing neural networks should address the following questions:

1. **When** to add new neurons? For instance, some methods (Liu et al., 2019; Kilcher et al., 2019) require the training loss to plateau before growing, whereas others grow using a predefined schedule.
2. **Where** to add new capacity? We can add new neurons to the existing layers or create new layers among the existing ones.
3. **How** to initialize the new capacity?

In this work we mainly focus on the question of **how** and introduce a new initialization method for the new neurons. Our approach can also be used to guide when and where to grow new neurons. However, in order to make our comparison with other initialization methods fair we keep the growing schedule (**where** and **when**) fixed.

Baseline: GradMax (Evci, et al., 2022) - Limitations

1. only addresses "how" to grow neural network, but does not answer "when" nor "where" to grow;



Baseline: GradMax (Evci, et al., 2022) - Limitations

1. only addresses "how" to grow neural network, but does not answer "when" nor "where" to grow;
2. simply sets W_l^{new} as zeros, while which is found to be critical not only in determining the activation of specific instances, but also controlling the gradient direction in subsequent updates;

Baseline: GradMax (Evci, et al., 2022) - Limitations

1. only addresses "how" to grow neural network, but does not answer "when" nor "where" to grow;
2. simply sets W_l^{new} as zeros, while which is found to be critical not only in determining the activation of specific instances, but also controlling the gradient direction in subsequent updates;
3. only focused on maximizing gradient norm without considering the gradient's direction;

Baseline: GradMax (Evci, et al., 2022) - Limitations

1. only addresses “how” to grow neural network, but does not answer “when” nor “where” to grow;
2. simply sets W_l^{new} as zeros, while which is found to be critical not only in determining the activation of specific instances, but also controlling the gradient direction in subsequent updates;
3. only focused on maximizing gradient norm without considering the gradient’s direction;
4. lacks the interpretability of the newly added nodes;

Baseline: GradMax (Evci, et al., 2022) - Limitations

1. only addresses "how" to grow neural network, but does not answer "when" nor "where" to grow;
2. simply sets W_l^{new} as zeros, while which is found to be critical not only in determining the activation of specific instances, but also controlling the gradient direction in subsequent updates;
3. only focused on maximizing gradient norm without considering the gradient's direction;
4. lacks the interpretability of the newly added nodes;
5. not always stable in convergence;

Baseline: GradMax (Evci, et al., 2022) - Limitations

1. only addresses "how" to grow neural network, but does not answer "when" nor "where" to grow;
2. simply sets W_l^{new} as zeros, while which is found to be critical not only in determining the activation of specific instances, but also controlling the gradient direction in subsequent updates;
3. only focused on maximizing gradient norm without considering the gradient's direction;
4. lacks the interpretability of the newly added nodes;
5. not always stable in convergence;
6. ...

Preliminaries - The Manifold Hypothesis

- DNNs can be seen as to approximate data distributions or decision boundaries by learning **manifolds** in high dimensional feature spaces (Bengio, Mesnil, et al., 2013; Bengio, Courville, and Vincent, 2013);

Preliminaries - The Manifold Hypothesis

- DNNs can be seen as to approximate data distributions or decision boundaries by learning **manifolds** in high dimensional feature spaces (Bengio, Mesnil, et al., 2013; Bengio, Courville, and Vincent, 2013);

Definition (manifold)

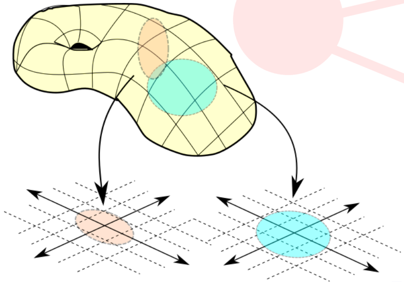
A manifold is a topological space that locally resembles Euclidean space near each point.

Preliminaries - The Manifold Hypothesis

- DNNs can be seen as to approximate data distributions or decision boundaries by learning **manifolds** in high dimensional feature spaces (Bengio, Mesnil, et al., 2013; Bengio, Courville, and Vincent, 2013);

Definition (manifold)

A manifold is a topological space that locally resembles Euclidean space near each point.



Preliminaries - MLE in Under-parameterized DNNs

- The training of DNNs mainly adopts the Maximum Likelihood Estimation (MLE).

Preliminaries - MLE in Under-parameterized DNNs

- The training of DNNs mainly adopts the Maximum Likelihood Estimation (MLE).

  Theorem 1

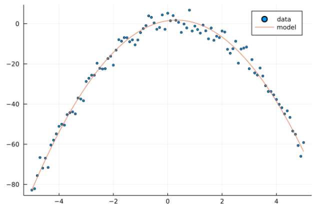
  When a neural network is small and less capable of approximation (lacking sufficient parameters to effectively capture the complexity of the datasets' domain distribution), it tends to converge to a best-fitted model where the majority of data can be correctly predicted.

Preliminaries - MLE in Under-parameterized DNNs

- The training of DNNs mainly adopts the Maximum Likelihood Estimation (MLE).

Theorem 1

When a neural network is small and less capable of approximation (lacking sufficient parameters to effectively capture the complexity of the datasets' domain distribution), it tends to converge to a best-fitted model where the majority of data can be correctly predicted.



Preliminaries - Deep ReLU Networks equal Shallow

- ReLU is one of the most standard activation functions in DNNs;

Preliminaries - Deep ReLU Networks equal Shallow

- ReLU is one of the most standard activation functions in DNNs;
- Recent studies (Hanin and Rolnick, 2019; Bietti and Bach, 2020; Mao and Zhou, 2023; Villani and Schoots, 2023) proved that:

Theorem 2

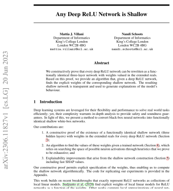
deep ReLU networks are equivalently replicable by corresponding shallow ReLU networks with only one hidden layer.

Preliminaries - Deep ReLU Networks equal Shallow

- ReLU is one of the most standard activation functions in DNNs;
- Recent studies (Hanin and Rolnick, 2019; Bietti and Bach, 2020; Mao and Zhou, 2023; Villani and Schoots, 2023) proved that:

Theorem 2

deep ReLU networks are equivalently replicable by corresponding shallow ReLU networks with only one hidden layer.



arXiv:2306.11827v1 [cs.LG] 20 Jun 2023

Any Deep ReLU Network is Shallow

Mattia J. Villani
Department of Informatics
King's College London
London WC2B 4BG
mattia.villani@kcl.ac.uk

Nandi Schoots
Department of Informatics
King's College London
London WC2B 4BG
nandi.schoots@kcl.ac.uk

Abstract

We constructively prove that every deep ReLU network can be rewritten as a functionally identical three-layer network with weights valued in the extended reals. Based on this proof, we provide an algorithm that, given a deep ReLU network, finds the explicit weights of the corresponding shallow network. The resulting shallow network is transparent and used to generate explanations of the model's behaviour.

1 Introduction

Deep learning systems are leveraged for their flexibility and performance to solve real world tasks efficiently; yet, their complexity warrants in-depth analysis to provide safety and soundness guarantees. In light of this, we present a method to convert black-box neural networks into functionally identical shallow white-box networks.

Our contributions are:

1. A constructive proof of the existence of a functionally identical shallow network (three hidden layers) with weights in the extended reals for every deep ReLU network (Section 3);
2. An algorithm to find the values of these weights given a trained network (Section 4), which relies on searching the space of possible neuron activations through heuristics that we prove to be exhaustive; and
3. Explainability improvements that arise from the shallow network construction (Section 5) including fast SHAP values.

Our constructive proof permits explicit specification of the weights, thus enabling us to compute the shallow network algorithmically. The code for replicating our experiments is provided in the Appendix.

This work builds on recent breakthroughs that exactly represent ReLU networks as collections of local linear models. Sudjianto et al. (2020) find explicit weights of local linear models for ReLU

Deep ReLU Networks Have Surprisingly Few Activation Patterns

Boris Hanin
Facebook AI Research
Texas A&M University
bhanin@math.tamu.edu

David Rolnick
University of Pennsylvania
Philadelphia, PA USA
drolnick@seas.upenn.edu

Abstract

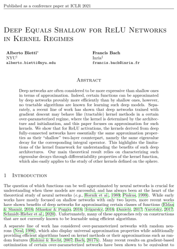
The success of deep networks has been attributed in part to their expressivity: per parameter, deep networks can approximate a richer class of functions than shallow networks. In ReLU networks, the number of activation patterns is one measure of expressivity; and the maximum number of patterns grows exponentially with the depth. However, recent work has showed that the practical expressivity of deep networks – the functions they can learn rather than express – is often far from the theoretical maximum. In this paper, we show that the average number of activation patterns for ReLU networks at initialization is bounded by the total number of neurons raised to the input dimension. We show empirically that this bound, which is independent of the depth, is tight both at initialization and during training, even on memorization tasks that should maximize the number of activation patterns. Our work suggests that realizing the full expressivity of deep networks may not be possible in practice, at least with current methods.

1 Introduction

A fundamental question in the theory of deep learning is why deeper networks often work better in practice than shallow ones. One proposed explanation is that, while even shallow neural networks are universal approximators [3, 7, 9, 16, 21], there are functions for which increased depth allows exponentially more efficient representations. This phenomenon has been quantified for various complexity measures [4, 5, 6, 10, 18, 19, 22, 23, 24, 25]. However, authors such as Ba and Caruana have called into question this point of view [2], observing that shallow networks can often be trained to imitate deep networks and thus that functions learned in practice by deep networks may not achieve the full expressive power of depth.

In this article, we attempt to capture the difference between the maximum complexity of deep networks and the complexity of functions that are actually learned (see Figure 1). We provide theoretical and empirical analyses of the typical complexity of the function computed by a ReLU network $\mathcal{N}$. Given a vector θ of its trainable parameters, $\mathcal{N}$ computes a continuous and piecewise linear function $x \mapsto \mathcal{N}(x;\theta)$. Each θ thus is associated with a partition of input space $\mathbb{R}^{n_{in}}$ into activation regions, polytopes on which $\mathcal{N}(x;\theta)$ computes a single linear function corresponding to a fixed activation pattern in the neurons of $\mathcal{N}$.

We aim to count the number of such activation regions. This number has been the subject of previous work (see §1.1), with the majority concerning large *lower bounds* on the *maximum* over all θ of the number of regions for a given network architecture. In contrast, we are interested in the typical behavior of ReLU nets as they are used in practice. We therefore focus on small *upper bounds* for the



Published as a conference paper at ICLR 2021

DEEP EQUALS SHALLOW FOR RELU NETWORKS IN KERNEL REGIMES

Alberto Bietti[*]
NYU[†]
alberto.bietti@nyu.edu

Francis Bach
Inria[‡]
francis.bach@inria.fr

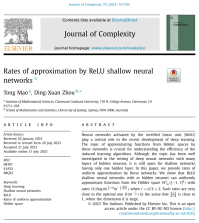
ABSTRACT

Deep networks are often considered to be more expressive than shallow ones in terms of approximation. Indeed, certain functions can be approximated by deep networks provably more efficiently than by shallow ones, however, no tractable algorithms are known for learning such deep models. Separately, a recent line of work has shown that deep networks trained with gradient descent may behave like (tractable) kernel methods in a certain over-parameterized regime, where the kernel is determined by the architecture and initialization, and this paper focuses on approximation for such kernels. We show that for ReLU activations, the kernels derived from deep fully-connected networks have essentially the same approximation properties as their "shallow" two-layer counterpart, namely the same eigenvalue decay for the corresponding integral operator. This highlights the limitations of the kernel framework for understanding the benefits of such deep architectures. Our main theoretical result relies on characterizing such eigenvalue decays through differentiability properties of the kernel function, which also easily applies to the study of other kernels defined on the sphere.

1 INTRODUCTION

The question of which functions can be well approximated by neural networks is crucial for understanding when these models are successful, and has always been at the heart of the theoretical study of neural networks (*e.g.*, Hornik et al., 1989; Pinkus, 1999). While early works have mostly focused on shallow networks with only two layers, more recent works have shown benefits of deep networks for approximating certain classes of functions (Eldan & Shamir, 2016; Mhaskar & Poggio, 2016; Telgarsky, 2016; Daniely, 2017; Yarotsky, 2017; Schmidt-Hieber et al., 2020). Unfortunately, many of these approaches rely on constructions that are not currently known to be learnable using efficient algorithms.

A separate line of work has considered over-parameterized networks with random neurons (Neal, 1996), which also display universal approximation properties while additionally providing efficient algorithms based on kernel methods or their approximations such as random features (Rahimi & Recht, 2007; Bach, 2017b). Many recent results on gradient-based optimization of certain over-parameterized networks have been shown to be equivalent to



Journal of Complexity 79 (2023) 101784

Contents lists available at ScienceDirect

Journal of Complexity

journal homepage: www.elsevier.com/locate/jco

ELSEVIER

Rates of approximation by ReLU shallow neural networks[☆]

Tong Mao[a], Ding-Xuan Zhou[b,*]

[a] Institute of Mathematical Sciences, Claremont Graduate University, 710 N. College Avenue, Claremont, CA 91711, USA
[b] School of Mathematics and Statistics, University of Sydney, Sydney, NSW 2006, Australia

ARTICLE INFO

Article history:
Received 30 January 2022
Received in revised form 20 July 2023
Accepted 21 July 2023
Available online 31 July 2023

MSC:
68T07
41A25
68Q32

Keywords:
Deep learning
Shallow neural networks
ReLU
Rates of uniform approximation
Hölder space

ABSTRACT

Neural networks activated by the rectified linear unit (ReLU) play a central role in the recent development of deep learning. The topic of approximating functions from Hölder spaces by these networks is crucial for understanding the efficiency of the induced learning algorithms. Although the topic has been well investigated in the setting of deep neural networks with many layers of hidden neurons, it is still open for shallow networks having only one hidden layer. In this paper, we provide rates of uniform approximation by these networks. We show that ReLU shallow neural networks with m hidden neurons can uniformly approximate functions from the Hölder space $W_\infty^r([-1,1]^d)$ with rates $O((\log m)^{\frac{1}{2}+d} m^{-\frac{r}{d}\frac{d+2}{d+4}})$ when $r < d/2+2$. Such rates are very close to the optimal one $O(m^{-\frac{r}{d}})$ in the sense that $\frac{d+2}{d+4}$ is close to 1, when the dimension d is large.

© 2023 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Preliminaries - Local Linearity of ReLU-based MLP

- Multilayer Perceptron (MLP) is the most standard DNN model;

Preliminaries - Local Linearity of ReLU-based MLP

- Multilayer Perceptron (MLP) is the most standard DNN model;
- It is easy to prove that:

Proposition 1

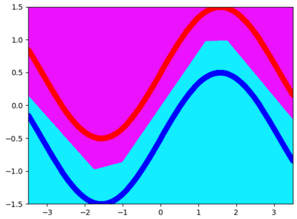
The manifolds approximated by ReLU-based MLP models are locally linear, consisting of multiple linear hyper-planes. Each linear segment corresponds to a different activation pattern of the ReLU units, giving the model its ability to approximate a locally linear behavior within each segment.

Preliminaries - Local Linearity of ReLU-based MLP

- Multilayer Perceptron (MLP) is the most standard DNN model;
- It is easy to prove that:

Proposition 1

The manifolds approximated by ReLU-based MLP models are locally linear, consisting of multiple linear hyper-planes. Each linear segment corresponds to a different activation pattern of the ReLU units, giving the model its ability to approximate a locally linear behavior within each segment.

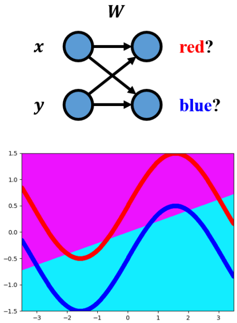


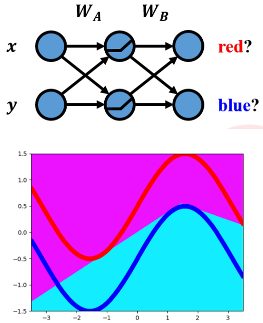
Preliminaries - Growing DNNs Increase Non-linearity

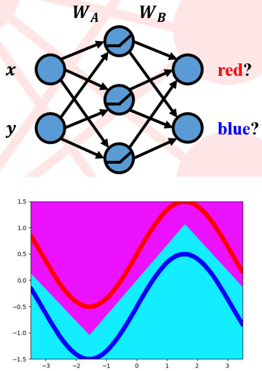
Observation 1

Growing a DNN by adding new node(s) to broaden a specific layer introduces new dimension(s) to the latent feature space associated with that layer, which is, in nature, increasing the non-linearity of the low-dimensional projections of the manifold to enhances the network's capacity to model complex patterns.

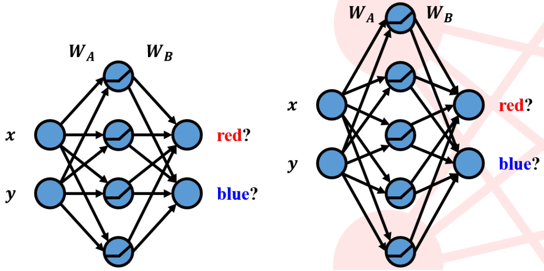
Preliminaries - Growing DNNs Increase Non-linearity

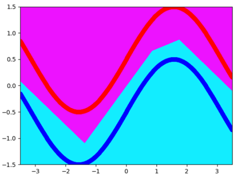
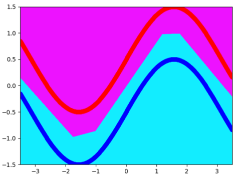






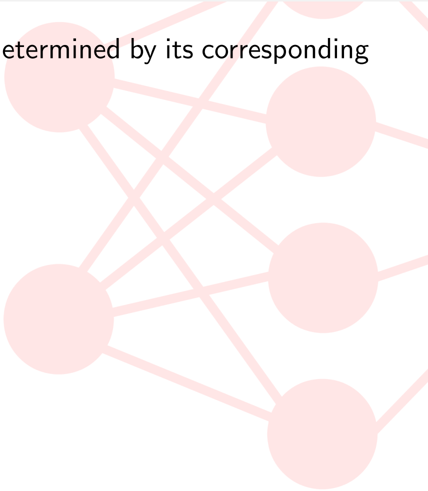
Preliminaries - Growing DNNs Increase Non-linearity



Key Ideas

1. Every decision on a single input is determined by its corresponding local linearity unit.



Key Ideas

1. Every decision on a single input is determined by its corresponding local linearity unit.
   - "activated linear segment"

Key Ideas

1. Every decision on a single input is determined by its corresponding local linearity unit.
    - "activated linear segment"
2. In a less-capable model, wrong predictions originate from either noise or significant patterns that play useful roles in making more generally correct predictions.

Key Ideas

1. Every decision on a single input is determined by its corresponding local linearity unit.
    - "activated linear segment"
2. In a less-capable model, wrong predictions originate from either noise or significant patterns that play useful roles in making more generally correct predictions.
    - "typical mis-predicted instance"

Key Ideas

1. Every decision on a single input is determined by its corresponding local linearity unit.
   - "activated linear segment"
2. In a less-capable model, wrong predictions originate from either noise or significant patterns that play useful roles in making more generally correct predictions.
   - "typical mis-predicted instance"
3. A less-capable model can be improved by enhancing its capability to handle each typical mis-predicted instance at a time.

Key Ideas

1. Every decision on a single input is determined by its corresponding local linearity unit.
    - "activated linear segment"
2. In a less-capable model, wrong predictions originate from either noise or significant patterns that play useful roles in making more generally correct predictions.
    - "typical mis-predicted instance"
3. A less-capable model can be improved by enhancing its capability to handle each typical mis-predicted instance at a time.
4. Specifically, the model's capability (i.e., approximatability) can be increased by introducing non-linearity to its complexity,

Key Ideas

1. Every decision on a single input is determined by its corresponding local linearity unit.
   - "activated linear segment"
2. In a less-capable model, wrong predictions originate from either noise or significant patterns that play useful roles in making more generally correct predictions.
   - "typical mis-predicted instance"
3. A less-capable model can be improved by enhancing its capability to handle each typical mis-predicted instance at a time.
4. Specifically, the model's capability (i.e., approximatability) can be increased by introducing non-linearity to its complexity,
5. which, can be achieved by breaking / folding the corresponding activated linear segment that leads to the typical mis-prediction.

Definitions

Definition (Activated Linear Segment)

For manifolds in high-dimensional spaces that are approximated by ReLU-based MLPs, their projections onto the input space are locally linear. In classification tasks, these projections are the decision boundaries, whereas in prediction tasks, they are the regression curves. Each piece of linear segments of these projections results from a composition of affine transformations governed by a specific activation pattern of ReLU units. These segments can thus be called "activated linear segments".

Definitions

Definition (Decision Boundary)

An $n-1$ dimensional hyper-plane on the n dimensional input space for distinguishing between two classes (the wrongly predicted class i given input and its ground truth class j), which is defined by equation $\vec{\boldsymbol{w}_{i-j}} \cdot \vec{\boldsymbol{x}} + b_{i-j} = 0$, where $\vec{\boldsymbol{w}_{i-j}}$ and b_{i-j} denote the differences between row i and row j of the affine transformation of the corresponding activated linear segment.

Definitions

Definition (Folding Crease)

An $n-2$ dimensional sub-space on the $n-1$ dimensional hyper-plane of the decision boundary that acts as the "transition line" between two contiguous activated linear segments, where the activation pattern changes.

Definition (Fold Point)

A single point $\vec{x_f}$ (an n dimensional vector) on the Folding Crease that can use for guiding the location of Folding Crease on the $n-1$ dimensional hyper-plane of the decision boundary:
$$\begin{cases} \vec{\boldsymbol{w}_{i-j}} \cdot \vec{\boldsymbol{x}} + b_{i-j} = 0 \\ \vec{\boldsymbol{u}} \cdot (\vec{\boldsymbol{x}} - \vec{\boldsymbol{x}_f}) = 0 \end{cases}$$
where $\vec{\boldsymbol{u}}$ is a non-zero n dimensional vector that is orthogonal to the normal vector of the $n-1$ dimensional hyper-plane of the decision boundary $\vec{\boldsymbol{w}_{i-j}}$, i.e. $\vec{\boldsymbol{u}} \cdot \vec{\boldsymbol{w}_{i-j}} = 0$

Definitions

Definition (Typical Mis-predicted Instance)

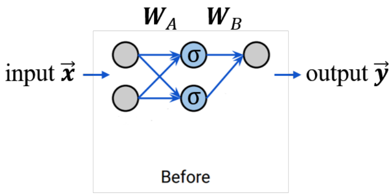
Among all input data that is wrongly predicted by an model, the single input data instance that can be represents a certain kind of mistakes that the model cannot distinguish is called a "typical mis-predicted instance".

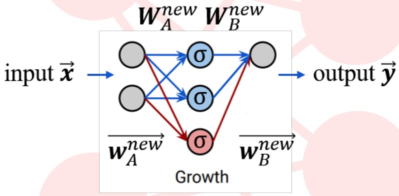
Determining Values for New Parameters

Theorem 3

Adding a Folding Crease on the Decision Boundary of an existing Acivated Linear Segment at a specific location $\vec{x_f}$ can be implemented by introducing a new node to the first layer of the shallow ReLU network and enfore the activation of its parameters (denoted as $\boldsymbol{w}_A^{\vec{new}}$ and b_A^{new} respectively) to occur exactly at point $\vec{x_f}$ using the equation $\boldsymbol{w}_A^{new} \cdot \vec{x_f} + b_A^{new} = 0$.

Notations & Symbols





$$W_A^{new} = \begin{bmatrix} W_A \\ \overrightarrow{w_A^{new}} \end{bmatrix}$$

$$W_B^{new} = \begin{bmatrix} W_B & \overrightarrow{w_B^{new}} \end{bmatrix}$$

Determining Values for New Parameters

Lamma 1

If and only if the $\boldsymbol{w}_A^{\vec{new}}$ can be represented as an linear combination of $\boldsymbol{w}_{i-j}^{\vec{}}$ and $\vec{\boldsymbol{u}}$ (i.e. there exist scalars α and β such that $\boldsymbol{w}_A^{\vec{new}} = \alpha \boldsymbol{w}_{i-j}^{\vec{}} + \beta \vec{\boldsymbol{u}}$), there is one and only one solution for variable b_A^{new} that for all input $\vec{\boldsymbol{x}}$ on the Folding Crease, satisfy the above equation $\boldsymbol{w}_A^{\vec{new}} \cdot \vec{\boldsymbol{x}} + b_A^{new} = 0$.

Determining Values for New Parameters

Lamma 2

$$\frac{d\mathcal{L}}{d\vec{\boldsymbol{x}}} = (\boldsymbol{W}_B \sigma(\cdot)' \boldsymbol{W}_A)^T \frac{d\mathcal{L}}{d\vec{\boldsymbol{y}}} + (\boldsymbol{w}_B^{\vec{new}} \cdot \frac{d\mathcal{L}}{d\vec{\boldsymbol{y}}})\boldsymbol{w}_A^{\vec{new}}$$

Theorem 4

$\boldsymbol{w}_A^{\vec{new}}$ directly affect the direction of $\frac{d\mathcal{L}}{d\vec{\boldsymbol{x}}}$ by specifying the direction of its component vector, while $\boldsymbol{w}_B^{\vec{new}}$ indirectly affect $\frac{d\mathcal{L}}{d\vec{\boldsymbol{x}}}$ by determining the norm of its component vector.

Determining Values for New Parameters

Lamma 3

$\boldsymbol{w}_A^{\vec{n}ew}$ directly determines which input domains can activate the newly added node

Theorem 5

$\boldsymbol{w}_A^{\vec{n}ew}$ determines the location of the fold point after adding a new node.

Determining Values for New Parameters

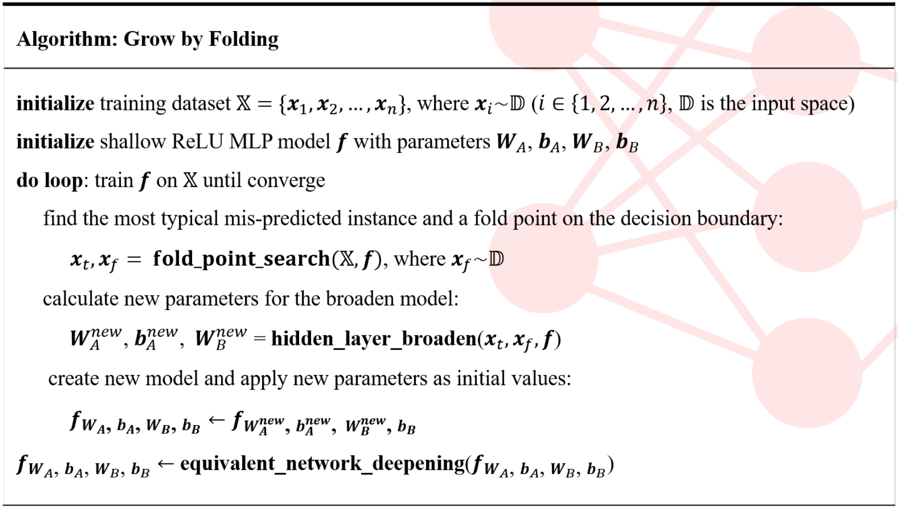
Lamma 4

To ensure the affine transformation of the Activated Linear Segment does not vary much before and after introducing the new node, the parameters of the newly added connection in the second layer of the network model (denoted as $\boldsymbol{w}_B^{\vec{new}}$) need to approach: $\begin{cases} \boldsymbol{w}_A^{\vec{new}} \otimes \boldsymbol{w}_B^{\vec{new}} \to \mathbf{0}_{n,n} \\ b_A^{new} \cdot \boldsymbol{w}_B^{\vec{new}} \to \vec{\mathbf{0}} \end{cases}$ which can be further simplified as: $\min \left\| \boldsymbol{w}_B^{new} \right\|^2$

Lamma 5

The above $\left\| \boldsymbol{w}_B^{new} \right\|^2$ cannot be exactly equal to zero, otherwise no gradient can be backprogagated to update the corresponding parameters $\boldsymbol{w}_A^{\vec{new}}$ and b_A^{new} in the first layer.

Proposed Method



Algorithm: Grow by Folding

```
initialize training dataset $\mathbb{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$, where $\boldsymbol{x}_i \sim \mathbb{D}$ ($i \in \{1, 2, \dots, n\}$, $\mathbb{D}$ is the input space)
initialize shallow ReLU MLP model $\boldsymbol{f}$ with parameters $\boldsymbol{W}_A$, $\boldsymbol{b}_A$, $\boldsymbol{W}_B$, $\boldsymbol{b}_B$
do loop: train $\boldsymbol{f}$ on $\mathbb{X}$ until converge
    find the most typical mis-predicted instance and a fold point on the decision boundary:
        $\boldsymbol{x}_t, \boldsymbol{x}_f = \textbf{fold\_point\_search}(\mathbb{X}, \boldsymbol{f})$, where $\boldsymbol{x}_f \sim \mathbb{D}$
    calculate new parameters for the broaden model:
        $\boldsymbol{W}_A^{new}, \boldsymbol{b}_A^{new}, \boldsymbol{W}_B^{new} = \textbf{hidden\_layer\_broaden}(\boldsymbol{x}_t, \boldsymbol{x}_f, \boldsymbol{f})$
    create new model and apply new parameters as initial values:
        $\boldsymbol{f}_{\boldsymbol{W}_A, \boldsymbol{b}_A, \boldsymbol{W}_B, \boldsymbol{b}_B} \leftarrow \boldsymbol{f}_{\boldsymbol{W}_A^{new}, \boldsymbol{b}_A^{new}, \boldsymbol{W}_B^{new}, \boldsymbol{b}_B}$
$\boldsymbol{f}_{\boldsymbol{W}_A, \boldsymbol{b}_A, \boldsymbol{W}_B, \boldsymbol{b}_B} \leftarrow \textbf{equivalent\_network\_deepening}(\boldsymbol{f}_{\boldsymbol{W}_A, \boldsymbol{b}_A, \boldsymbol{W}_B, \boldsymbol{b}_B})$
```

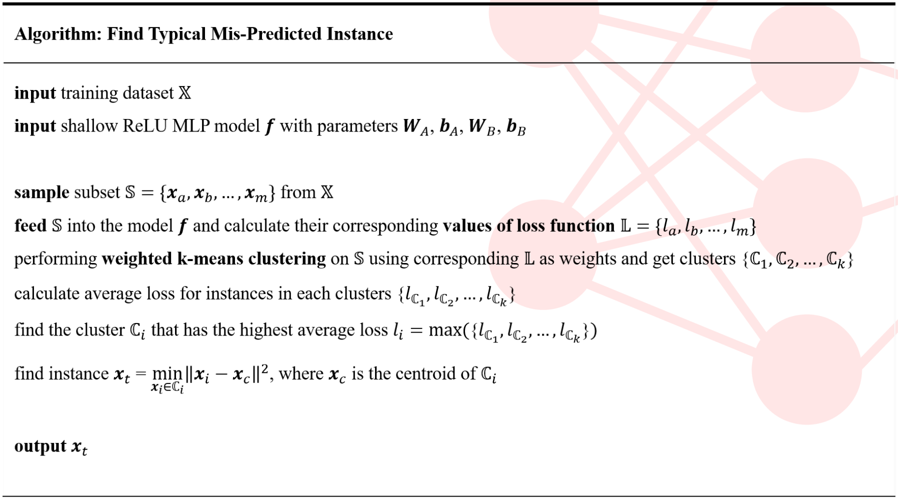
Proposed Method

Algorithm: Fold Point Search

```
input training dataset $\mathbb{X}$
input shallow ReLU MLP model $\boldsymbol{f}$ with parameters $\boldsymbol{W}_A, \boldsymbol{b}_A, \boldsymbol{W}_B, \boldsymbol{b}_B$

find the most typical mis-predicted instance:
    $\boldsymbol{x}_t = $ find_typical_mis_predicted_instance($\mathbb{X}, \boldsymbol{f}$)
find the activated linear segment of model $\boldsymbol{f}$ given $\boldsymbol{x}_t$ as input:
    $\boldsymbol{W}_{activated}, \boldsymbol{b}_{activated} = $ find_activate_linear_segment($\boldsymbol{x}_t, \boldsymbol{W}_A, \boldsymbol{b}_A, \boldsymbol{W}_B, \boldsymbol{b}_B$)
decision boundary regarding $\boldsymbol{x}_t$ can be defined by:
    $\{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_\infty\}$ s.t. $\boldsymbol{W}_{activated}\boldsymbol{x}_i + \boldsymbol{b}_{activated} = \boldsymbol{c}$, where $\boldsymbol{c}$ is a uniform vector
find the fold point $\boldsymbol{x}_f$ on the decision boundary that is closest to $\boldsymbol{x}_t$:
    $$\boldsymbol{x}_f = \min_{\boldsymbol{d} \in \{\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_\infty\}} \|\boldsymbol{d} - \boldsymbol{x}_t\|^2$$
output $\boldsymbol{x}_t, \boldsymbol{x}_f$
```

Proposed Method



Algorithm: Find Typical Mis-Predicted Instance

```
input training dataset $\mathbb{X}$
input shallow ReLU MLP model $\boldsymbol{f}$ with parameters $\boldsymbol{W}_A, \boldsymbol{b}_A, \boldsymbol{W}_B, \boldsymbol{b}_B$

sample subset $\mathbb{S} = \{\boldsymbol{x}_a, \boldsymbol{x}_b, \dots, \boldsymbol{x}_m\}$ from $\mathbb{X}$
feed $\mathbb{S}$ into the model $\boldsymbol{f}$ and calculate their corresponding values of loss function $\mathbb{L} = \{l_a, l_b, \dots, l_m\}$
performing weighted k-means clustering on $\mathbb{S}$ using corresponding $\mathbb{L}$ as weights and get clusters $\{\mathbb{C}_1, \mathbb{C}_2, \dots, \mathbb{C}_k\}$
calculate average loss for instances in each clusters $\{l_{\mathbb{C}_1}, l_{\mathbb{C}_2}, \dots, l_{\mathbb{C}_k}\}$
find the cluster $\mathbb{C}_i$ that has the highest average loss $l_i = \max(\{l_{\mathbb{C}_1}, l_{\mathbb{C}_2}, \dots, l_{\mathbb{C}_k}\})$
find instance $\boldsymbol{x}_t = \min_{\boldsymbol{x}_i \in \mathbb{C}_i} \|\boldsymbol{x}_i - \boldsymbol{x}_c\|^2$, where $\boldsymbol{x}_c$ is the centroid of $\mathbb{C}_i$

output $\boldsymbol{x}_t$
```

Proposed Method

Algorithm: Hidden Layer Broaden

```
input typical mis-predicted point $x_t$
input fold point on the decision boundary $x_f$
input shallow ReLU MLP model $f_{W_A,\, b_A,\, W_B,\, b_B}$ with parameters $W_A, b_A, W_B, b_B$
ensure $\forall x$: $\boldsymbol{W}_B^{new}\boldsymbol{W}_A^{new}x = \boldsymbol{W}_B\boldsymbol{W}_A x$:
    set $\boldsymbol{W}_B^{new} = [\boldsymbol{W}_B \quad \boldsymbol{0}]$
    set $\boldsymbol{b}_A^{new} = \begin{bmatrix} \boldsymbol{b}_A \\ 0 \end{bmatrix}$
    set $\boldsymbol{W}_A^{new} = \begin{bmatrix} \boldsymbol{W}_A \\ w_A^{+} \end{bmatrix}$ s.t.:
        ensure the grown new model fold only at point $x_f$:
            $$\boldsymbol{W}_A^{new} x_f + \boldsymbol{b}_A^{new} = \boldsymbol{0}$$
        apply GradMax to $\boldsymbol{W}_A^{new}$
            $$\max_{\boldsymbol{W}_A^{new}} \|\mathbb{E}_{\mathbb{D}} \nabla \boldsymbol{W}_A^{new}\|_F^2$$
        align gradient direction with $\boldsymbol{W}_A^{new}(x_t - x_f)$
            $$\max_{\boldsymbol{W}_A^{new}} \frac{\nabla \boldsymbol{W}_A^{new} x_t \cdot \boldsymbol{W}_A^{new}(x_t - x_f)}{\|\nabla \boldsymbol{W}_A^{new} x_t\| \cdot \|\boldsymbol{W}_A^{new}(x_t - x_f)\|}$$
output $\boldsymbol{W}_A^{new}$, $\boldsymbol{b}_A^{new}$, $\boldsymbol{W}_B^{new}$
```

Supplementary Details

1. "Dead Layer" Problem

Supplementary Details

1. "Dead Layer" Problem
    - Batch Normalization

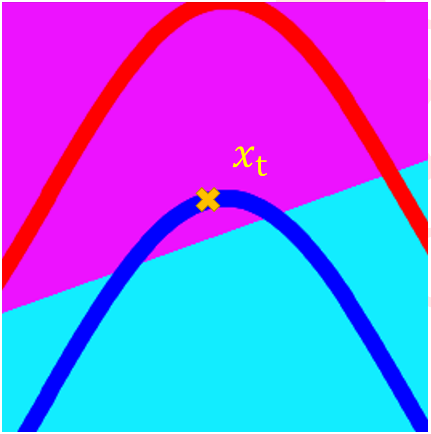
Supplementary Details

1. "Dead Layer" Problem
   - Batch Normalization
2. Optimization Problem of Fold Point Search Algorithm

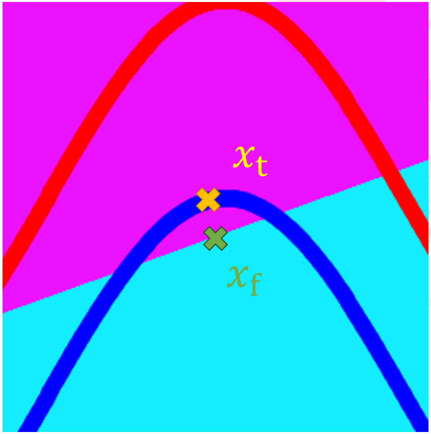
Supplementary Details

1. "Dead Layer" Problem
   - Batch Normalization
2. Optimization Problem of Fold Point Search Algorithm
   - given the equation of decision bondary: $w \cdot x + b = 0$, the closest point on the decision boundary can be found by $x_t - ((w \cdot x_t) + b) * (w/(w \cdot w))$

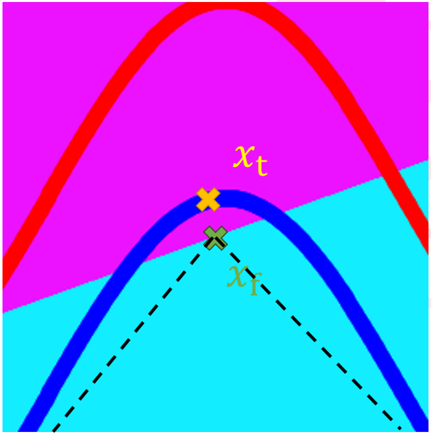
Example



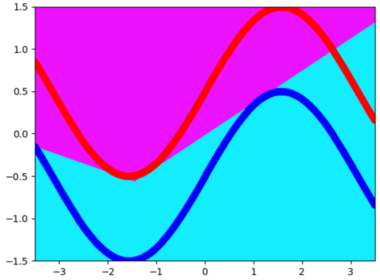
Example

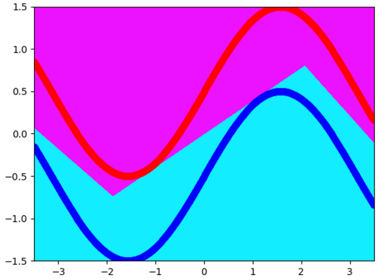


Example

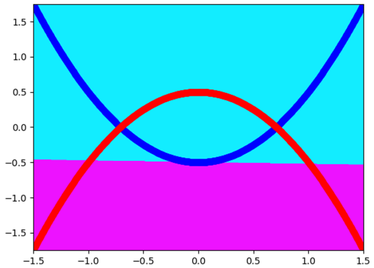


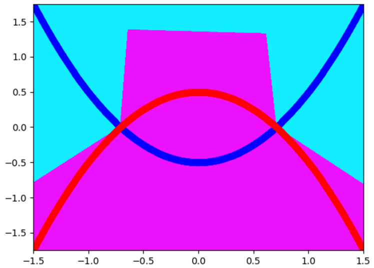
Binary Classification: Sine



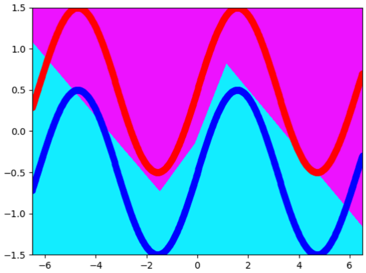


Binary Classification: Overlap Quadratic





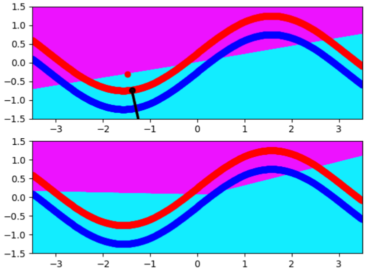
Utilization of Approximatability



- random initialized: **65** / 1000
- grow from 2-2-2 to 2-3-2: **360** / 1000

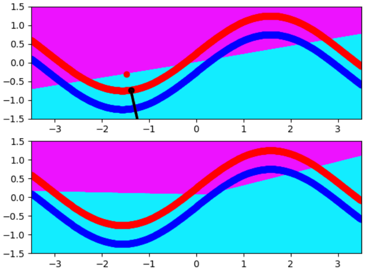
Next Milestones

1. due to the difference in updating speeds between the old and new parameters, the proposed direction along $\vec{x_t} - \vec{x_f}$ is not the most ideal gradient direction for updates



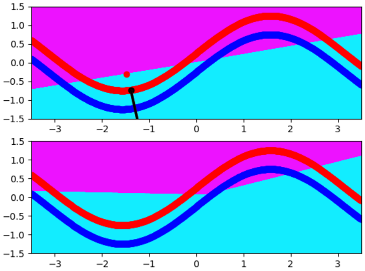
Next Milestones

1. due to the difference in updating speeds between the old and new parameters, the proposed direction along $\vec{x_t} - \vec{x_f}$ is not the most ideal gradient direction for updates
2. Folding point does not lie on the effective activated linear segment



Next Milestones

1. due to the difference in updating speeds between the old and new parameters, the proposed direction along $\vec{x_t} - \vec{x_f}$ is not the most ideal gradient direction for updates
2. Folding point does not lie on the effective activated linear segment
3. Constrainted Optimization Difficulty



References I

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, *35*(8), 1798–1828.

Bengio, Y., Mesnil, G., Dauphin, Y., & Rifai, S. (2013). Better mixing via deep representations. *International conference on machine learning*, 552–560.

Bietti, A., & Bach, F. (2020). Deep equals shallow for relu networks in kernel regimes. *arXiv preprint arXiv:2009.14397*.

Cheng, H., Zhang, M., & Shi, J. Q. (2024). A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, *20*(55), 1–21.

References II

Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: From architectures to learning. *Evolutionary intelligence*, *1*, 47–62.

Galván, E., & Mooney, P. (2021). Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence*, *2*(6), 476–493.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2021). Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(11), 7436–7456.

Hanin, B., & Rolnick, D. (2019). Deep relu networks have surprisingly few activation patterns. *Advances in neural information processing systems*, *32*.

Mao, T., & Zhou, D.-X. (2023). Rates of approximation by relu shallow neural networks. *Journal of Complexity*, *79*, 101784.

References III

Villani, M. J., & Schoots, N. (2023). Any deep relu network is shallow. *arXiv preprint arXiv:2306.11827.*

Thank you!